A Composite Agent Architecture for Multi-Agent Simulations

Michael VanPutte
Brian Osborn
John Hiles
MOVES Institute
Naval Postgraduate School
1 University Circle
Monterey, CA 93943-5118
831-656-4074, 831-656-3733,831- 656-2988
michael.vanputte@us.army.mil, baosborn@nps.navy.mil, jehiles@nps.navy.mil

Keywords:
multi-agent systems, multi-agent simulations, information assurance

ABSTRACT: *The MOVES Institute's Computer-Generated Autonomy Group has focused on a research goal of modeling complex and adaptive behavior while at the same time making the behavior easier to create and control. This research has led to several techniques for agent construction, that includes a social and organization relationship management engine, a composite agent architecture, an agent goal apparatus, a structure for capturing and applying procedural knowledge (tickets), and the ability to bring these technologies to bear at the right time and in the proper context through connectors. This paper provides an overview of the architecture and discusses the implementation of this architecture in a multi-agent simulation of the information assurance domain.*

1. Introduction

In 1999 the Naval Postgraduate School MOVES (Modeling, Virtual Environment, and Simulation) Institute added a new research direction in the area of multi-agent systems and computer generated autonomous behavior. From the outset, MOVES agent research has had two goals. First, to bring rich, complex, adaptive behavior to Department of Defense (DoD) related models, simulations and other systems through the application of multi-agent technology. And second, to make this adaptive behavior far easier to achieve and control. This latter characteristic will allow problem solvers to focus their attention and intellect on the agent's problem solving behavior and not on the implementation mechanism. The intent is to shift the focus from "how do we do this?" to "what can we do with this?"

This paper describes several innovations in the field of agent-based system simulation using information assurance as the domain of discourse and introduces two new research areas being investigated in the MOVES Institute.

Portions of this paper originally appeared in [1].

2. Semi-Fluid Software Structure and Emergent Behavior

2.1 Introduction

Software development has traditionally focused on building software based on rigidly structured architectures with terms like "structure" and "architecture" usually referring to fixed and immutable relationships among the components inside the software. Many in the computer science and software engineering community assume structure must be rigid and tightly bound at design time if a program has any chance of meeting its design goals. This outlook is analogous to our view of a new highway system that is designed on paper and constructed with concrete and steel to meet the forecast needs of a growing city. Once built, the highway system remains fixed and static unless new construction occurs. It would be absurd to expect it to mold itself into new forms to meet growing infrastructure and changing traffic patterns. This same thinking has held true for traditional software designs. The architecture is fixed at design time; its structure is inert.

The study of computer generated autonomous behavior is supplementing this thinking by exploring the use of multi-agent systems (MAS) to build software that modifies its own structure, within a set of constraints, to maintain close contact with a dynamic environment.

MAS research at the MOVES Institute is founded on the premise that semi-fluid software structures are not only possible, but essential to developing truly adaptive simulations and modeling emergent behavior.

2.2 A Design Paradigm Shift

A real challenge when first encountering multi-agent system simulations is coming to grips with emergent behavior in software. Traditional problem solving in software engineering is *direct* in the sense that the developer conceives of an algorithmic solution and transfers that solution to software. Software development rigor and practice is used to insure the code will produce an exact execution of the algorithm. In direct solutions, the programmer knows exactly how to solve the problem and the software implements that solution precisely. This approach is fine for problems where the domain is well known, and the relationships are static, finite and well defined. Direct solution systems are somewhat analogous to well-behaved functions. For a given input, the designer knows what to expect for the output. Surprises are a clear indication of a bug in the system.

In sharp contrast, surprises in MAS simulations are not only okay, but are the desired end, as long as the system operates within boundaries that are explicitly determined. The software is intended to surprise the designer within a system of constraints! This is possible through the use of software agents that discover an *indirect* path to the solution, thereby allowing for the possibility of arriving at a solution the designer may not have previously considered. In this way, multi-agent systems are capable of producing innovative solutions. In the field of information assurance (IA), the relationships between the human actors and the technological components are dynamic and complex. The ability to achieve malicious goals is often an 'out-of-the-box' application of existing capabilities. An innovative MAS system is exactly the right tool for exploring the IA domain.

3. Information Assurance

Information Assurance is concerned with "…protect(ing) and defend(ing) information and information *systems* by ensuring their availability, integrity, authentication, confidentiality, and non-repudiation" [2]. The *system* is not restricted to technological components; it includes human actors that interact with the technical components. IA deals with adaptable humans and computational devices that are interconnected through webs of communications networks. Software and devices adapt, both autonomously and through human interaction, to perform tasks. Humans adapt themselves, communication links, devices, and the software running on those devices (sometime unknowingly) to better achieve their goals. The domain is a highly connected, dynamic environment, where small changes in one part of the environment can have tremendous, cascading effects in other parts.

This paper introduces a multi-agent simulation that is an implementation of a computational model of IA [3]. MARIA (Multi-Agent Research in Information Assurance) implements five agent-based system simulation innovations and provides an environment where investigators can conduct research and gain insight on information assurance.

4. Innovations in Agent Research

The Computer-Generated Autonomy Group has developed and refined five key techniques that further the research goal of making far more complex and adaptive behavior easier to create and control. The techniques include a social and organizational relationship management engine, a composite agent architecture, an agent goal apparatus, a structure for capturing and applying procedural knowledge (*tickets*), and the ability to bring these technologies to bear at the right time and in the proper context through *connectors*.

4.1 Social and Organizational Relationship Management Engine

The modeling and simulation community is continually being challenged to create rich, detailed models of ill-defined problems. Many of these problems are complex because of the involvement of human decision-making and organizational behavior. Humans and organizations have multiple levels of internal roles, goals and responsibilities, frequently conflicting with each other. While contemplating almost any decision, humans must evaluate a myriad of goals that they are currently attempting to achieve. These goals are sometimes supportive of each other, but often they are in conflict. Developing simulations that are capable of capturing this complex, often unpredictable, behavior is essential to realistically modeling large organizations accurately.

In an effort to simplify the development of MAS simulations and ease the integration of software agents into existing simulations, an agent modeling architecture called RELATE was created [4]. RELATE is an agent architecture for organizing agents into relationships, and allowing for functional specialization. The RELATE design paradigm focuses on the relationships between individuals and within organizations. By taking a relation-centric view of the problem domain the developer is encouraged to identify the various roles that are assumed by members belonging to each relationship. These roles have certain responsibilities and commitments, which tend to be manifested as additional goals that must be addressed by the various members of the relationship. Once an agent is a member of a relationship, it must base its action selection on its personality, its particular concern for each goal, and the state of achievement of each goal. Entering into a relationship connects or *binds* agents to one another, resulting in the assignment of new roles, goals and responsibilities. Relationships are often formed to achieve something that is not achievable by any one individual. In this way, agents can take advantage of shared resources and capabilities to achieve a goal that would otherwise be unattainable.

RELATE focuses the designer on six key concepts of MAS simulations: relationships, environment, laws, agents, things (objects), and effectors. A library of Java classes was developed that enabled the researcher to rapidly prototype an agent-based simulation, supporting cross-platform and web-based designs.

In MARIA, researchers declare *Organizations*. An organization is a collection of actor roles and organizational information assurance policies. Organizations range from formal enterprises (commercial and government entities), to informal collections of individuals with a common goal (hacker clubs, social groups, etc). The organization may represent a team (heterogeneous, interdependent roles) or a group (homogeneous, interchangeable roles) [5].

Roles are placeholders, initially defined but unfilled by actors, and represent a collection of behaviors specified for an organization. Some of the typical roles critical to an IA simulation are system users, system administrators, managers, cyber attackers, and vendors.

A role consists of *prerequisite Role Requirements*, a set of *Role Goals*, and *Tokens*. Role Goals are desires an agent pursues. Actors who commit to a role are given goals that are then added to the actor's goal set.

Roles have requirements that must be met prior to assuming a role. These prerequisites may be objects, prerequisite roles, or some particular actor capability or personality set attribute. Roles may also have corequisites that must be maintained. Failure to maintain corequisites could result in the role being revoked by the organization (being fired, thrown out of a group, etc.)

Tokens are abstract representations of objects that agents possess or require.

4.2 Composite Agents

Multi-agent system simulations typically consist of numerous high-level agents that represent entities operating in a common, shared environment. The agents residing in this shared environment, referred to as the "outer environment", interact with one another and the objects in the environment. They sense their environment, interpret the sensory input and make decisions as to what actions to take. These actions in turn affect the environment either directly through agent-to-environment interactions or indirectly through agent-to-agent interaction. In an effort to capture the strengths of both cognitive and reactive agents, while at the same time simplifying the design of such a complex agent, a *Composite Agent* architecture has been developed.

Composite Agents (CA) are composed of combinations of cognitive and reactive agents (Figure 4.1). They contain a set of cognitive *Symbolic Constructor Agents* (SCAs) that work with sensory streams (or *impressions*) from the outer environment to create a symbolic inner environment (E_{inner}) representing the agent's perspective of the outer environment (E_{outer}). The SCAs define the agent's sensor capabilities and are tailored to sense specific aspects of the environment. They also act to control and filter *impressions* of the outer environment, so the agent isn't overwhelmed in a rich outer environment. E_{inner} is influenced not only by what the SCAs sense, but also by the CA's internal state. For instance, in a predator-prey simulation, if the predator is hungry and senses an animal, it would show up in E_{inner} as food. On the other hand, if the predator has just eaten, then the animal would appear as just another animal in E_{inner}.

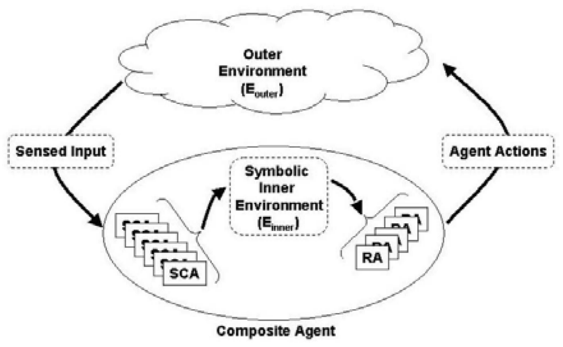
Outer Environment (E_{outer})
Sensed Input
Agent Actions
Symbolic Inner Environment (E_{inner})
SCA
RA
Composite Agent

Figure 4.1 A Composite Agent

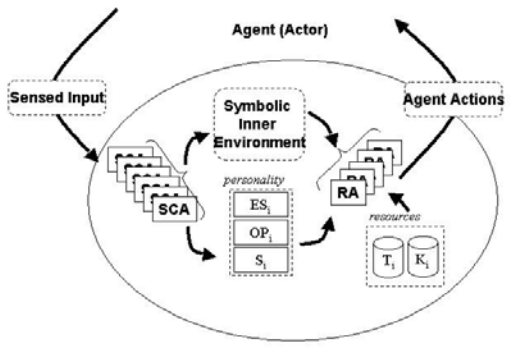

Figure 4.2 A MARIA Composite Agent

The symbolic inner environment is the agent's perception of the shared outer environment within which it operates. E_{inner} has little resemblance to the actual outer environment, rather it is an encoding of E_{outer} optimized to suit the Composite Agent's specific function. The role of an SCA is not unlike the role of radio navigation aid used by a pilot. The navigation aid senses radio signals in the outer environment and converts them into directional information that the pilot can use to navigate the aircraft. The inner environment used by the pilot for making decisions has little resemblance to the view looking out the window, but it is optimized for use by the pilot in navigating the aircraft.

Combined with the SCAs is a set of Reactive Agents that operate on the symbolic inner environment and generate actions for the CA to perform. Each RA has a set of possible goals and an apparatus for managing the process of selecting the active goal or goals.

In MARIA there are two subclasses of agents; actors (representing people) and infrastructures (representing an organization's information resources and processing capabilities).

A MARIA actor is a modified composite-agent (Figure 4.2). An actor consists of

- Sensor set – Set of SCA's
- Role set – Set of RA's
- T_i – Token set (resources and access rights)
- K_i –knowledge set
- ES_i – emotional state,
- OP_i – observable personality
- S_i – skill state

The Role set is the set of all Organizational Roles to which an agent has committed. This is implemented as a set of RAs, where each RA represents a role. An implied property for the actor is the set of all associated Goals SG_i. This is a set of all goals from all roles assigned to an actor. The set of goals an actor possesses from the numerous roles he is assuming defines the actor's behavior.

These goals are prioritized based on the actor's internal state (emotional, personality, and skill attributes) and the perception of the environment. Actors choose high priority goals and pursue them until they are completed, another goal receives a higher priority, or the agent decides the goal is no longer achievable.

The knowledge set K_i represents procedural problem solving capabilities. This knowledge base provides the actor with procedures to be used to achieve a goal.

The Actor's tokens T_i represent the collection of all objects the actor has collected.

The Personality state determines the actors' commitment and dedication for each goal. These are used by the goal apparatus to personalize the agent's goal prioritizing – thus creating outwardly observable differences in actor behavior.

An Actor's emotional state consists of a set of attributes that are an actor's current internal condition or feelings at any instant in time. These states represent a subset of the individual's emotions. These attributes may include the agent's feeling of loneliness, security, self-worth, and excitement.

The actor's Observable Personality values represent a relatively static set that defines that actor's long-term behavior. These values include propensities for risk, loyalty to organizations, ethics, etc.

Skills represent an abstract set of ability values the actors possess. These skills may include organizational technical skills, social, information technology, security, or management skills for example.

The architecture, when combined with the other agent components, facilitates the creation of complex agent behavior through relatively simple components.

4.3 Reactive Agents and Goal Management

Composite Agents contain numerous *Reactive Agents* (RAs), where each reactive agent is responsible for promoting a specific behavior of the *Composite Agent*. The set of RAs taken as a group, define the *Composite Agent's* set of high-level behaviors. The RAs operate within the world of the inner environment. They take as input sensory information from E_{inner}, and produce as output actions for the agent to perform.

Each RA has one or more goals specific to furthering the RA's behavior or function. So at any given time there are numerous goals competing for the *Composite Agent's* attention. Just as humans have multiple goals (sometimes conflicting), an agent too has multiple goals it wishes to satisfy. In human decision-making, goals are constantly shifting in priority, based on the person's context and state. Agents can mimic the flexibility and substitution skills of human decision-making through the use of a variable goal management apparatus within the RAs. It is from this goal apparatus where contextually appropriate, intelligent behavior emerges. RAs interpret the symbolic inner environment and through their goal apparatus, process this information to balance their goals and return an appropriate action for attaining their highest priority goal or goals (Figure 4.3).

Goals have four components; a state, a measurement method, a weight, and action or set of actions for achieving the goal. The goal's state is an indication of whether a goal is in an active, inactive, or some other domain specific state. The measurement method translates the sensory input received by the RA into a quantifiable measure of the current strength of a goal and how well it is being satisfied. This permits an agent to prioritize goals and adjust goal states based on context. A goal may also have a weight attached that can be used to adjust the importance or priority of the goal based on experience. Tied to each goal is an action or set of actions for achieving the goals under varying circumstances. The end result is that within the RA goal apparatus there are multiple goals that are constantly changing -- moving up and down -- with the top (active) goals dominating the agent and its behavior.

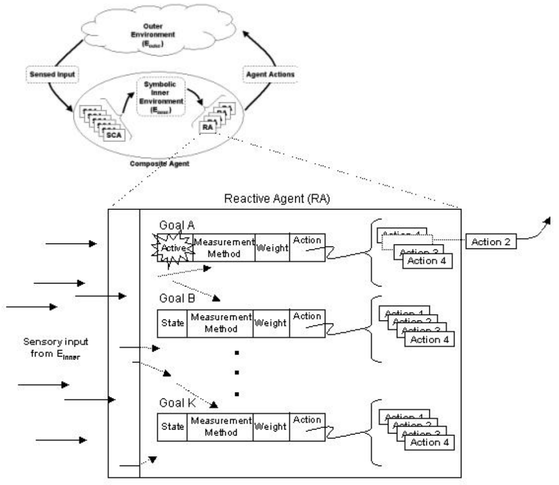

Figure 4.3 Reactive Agent

Additionally, agents can discard behaviors that do not further their goals, and increase the use of behaviors that have proved successful in reaching goals. This simple behavior serves as a reactive learning system where the agent learns from the environment, based on "what works" with no human expertise or intervention.

Goal switching based on a dynamically changing environment produces innovative and adaptive behavior, however, it is desirable to balance this with doctrinally correct and appropriate actions. This balance is achieved through the encoding of procedural knowledge in a data structure called *tickets*.

4.4 Tickets

Symbolic Constructor Agents and the goal apparatus were developed to control the agent's sensory capability and decision-making. In order to provide agents with rich procedurally oriented knowledge while still supporting adaptive behavior the agents knowledge base and action set has been encoded in a data structure called tickets. Tickets allow reactive agents to apply procedural knowledge in context. They define the agent's action set, i.e., its means to achieve its goals. They are used to organize procedural knowledge and provide the ability to balance doctrinal behavior with adaptive, innovative action, resulting in enriched problem solving behavior.

Tied to each of an agent's goals are one or more tickets that define how to achieve the goals. The tickets may have prerequisites or co-requisites that must be met in order for a ticket to be active (see *connectors* below). Additionally, tickets are composed of one or more frames, with each frame being one or more actions or behaviors. Various types of tickets have been defined, with choices ranging from uninterruptible to interruptible, and sequential to non-sequential.

Simply encoding procedural knowledge and linking it to various goals is not sufficient for creating intelligent behavior. The desire is to apply the most appropriate procedures for a given situation. The problem is that in a dynamic system the "given situation" not only changes constantly, but also is so complex, the system designer can't conceive of and account for every possibility. Therefore, the mechanism for determining the "most appropriate" procedures must be flexible and able to support the same level of complexity as the changing contexts of the dynamic system. The ability to take the correct action to match the situation is provided through the use of an apparatus called *connectors*.

MARIA actors possess two types of tickets; goal tickets, and knowledge tickets. Goal tickets are relatively static procedural steps that are bound to goals at compile-time. Knowledge tickets reside in the Knowledge set (K_i). These tickets bind to goals and other tickets at run-time, creating dynamic, unpredictable, yet appropriate behavior. This run-time binding is also performed through *connectors*.

4.5. Connectors

Connectors represent work that is based on symbolic types. They permit logical substitutions and sequencing, and facilitate explanations of reasoning. Connectors are a way to associate impressions, ideas and actions with a given context and achieve a logical sequence of behavior. Connectors are active objects that sense and react to the environment. They activate (extend) and deactivate (retract) based on the current context. As the agent's state and the state of the environment changes, the connectors sense the changes and extend or retract accordingly. By attaching connectors to various elements within the system, including tickets, the connectors signal the elements state of readiness and level of fitness for the current situation. With the connectors continually reacting to the environment, behavioral and procedural knowledge (tickets) can bind at runtime to fit the context as it develops. This binding is based not only on the state of the environment, but also on the goals of the agent and its social interactions with other agents. In this way, the correct procedural knowledge can be brought to bear in the correct situation.

In MARIA connectors react to operations performed by actors and infrastructure and have a potential for affecting other actors and infrastructures.

Connectors, as implemented in MARIA, are defined by the tuple {label, state, cardinality}.

4.5.1 Connector State – Extended or Retracted

Connectors have a Boolean state; extended or retracted. A retracted connector is inactive, and cannot connect to any other connector. An extended connector is currently available for connecting. If a connection occurs, and one of the connectors subsequently retracts, the binding is broken, and the remaining extended connector may bind to another extended connector. An extended connector can be distinguished from a retracted connector graphically by a small perpendicular tick on the retraced connector (Figure 4.4).

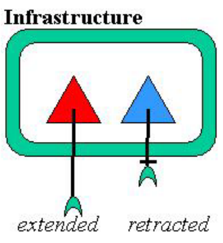

Figure 4.4 Extended and retracted connectors

Connectors are extended and retracted by actors and infrastructures to advertise services or request access to services. When an infrastructure wishes to advertise that it has a capability, it extends a socket connector. When an actor requests a resource, he extends a plug connector. If a socket accepts a plug, the two connectors are said to *bind.*

Connectors can extend without the owner of the connection being aware of this event. This 'hidden' connector can represent functionality on an infrastructure for instance, that is not an advertised capability. A buffer overflow vulnerability on a server could be represented as a 'hidden' socket connector, with special requirements to indicate knowledge of the vulnerability, and skills required to exploit the vulnerability.

4.5.2 Connector Labels and Ends

There are two types of connector ends: sockets and plugs. Sockets represent processes that can be utilized to access resources – a means to access information. When an agent requires a service or resource, he extends a plug connector and requests to bind to a socket. If a socket exists that matches the plug then the requesting agent *binds* to the resource or service.

Socket labels differ from plug labels. The tokens listed on a socket (T_{socket}) are the required tokens that *must* be presented to bind to this socket. The tokens listed on a plug (T_{plug}) are the tokens available to the owner of the plug. A binding will not occur unless $T_{socket} \subseteq T_{plug}$ (Figure 4.5).

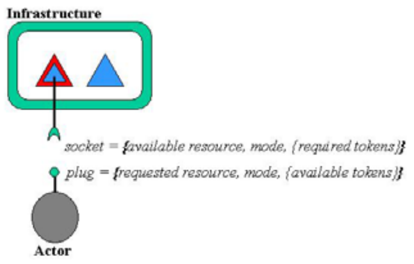
Infrastructure
socket = {available resource, mode, {required tokens}}
plug = {requested resource, mode, {available tokens}}
Actor

Figure 4.5 Actor binding to an Infrastructure

4.5.3 Connector Cardinality

Connectors have a cardinality that specifies the number of connectors that can simultaneously be bound to this particular connector. A connector without a cardinality label has a cardinality of one. A connector with a cardinality of zero represents a special type of connector called a Listener Connector (Figure 4.6).

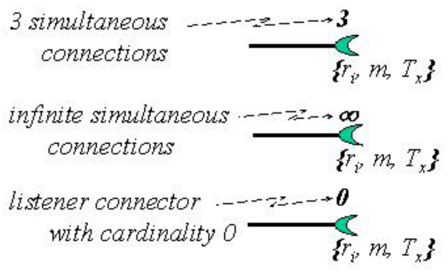
3 simultaneous connections
3
$\{r_i, m, T_x\}$
infinite simultaneous connections
∞
$\{r_i, m, T_x\}$
listener connector with cardinality 0
0
$\{r_i, m, T_x\}$

Figure 4.6 Socket Cardinality

Connectors are a powerful tool that binds the components of the simulation together. Properties of agents connect to tickets and goals to activate and deactivate these activities. Agents extend and retract connections to advertise and use services of other agents.

Furthermore, connector links can be traversed to explain agent reasoning. These connector-based components facilitate rapid development of modular, connector-based simulations.

4.6 MARIA

MARIA was developed as a proof of principle implementation of these technologies, with the goal of modeling the IA domain. The simulation allows researchers to rapidly create scenarios, and investigate the results of actor's actions and inactions as they pertain to an organization's information security. The composite architecture, combined with the connector-based simulation provides a modular system for modeling a complex domain.

5. MOVES Agent Research: What's Ahead

The multi-generational MAS research and insight gained over the past three years has manifested itself in increasingly complex simulations that were progressively easier to design and implement. This progress has allowed the MOVES Computer-Generated Autonomy Group to branch off into some very diverse areas of research.

5.1. Computer Generated Interactive Stories

These research projects represent exciting new directions for the MOVES Institute. The domains include interactive story generation and agent-based simulation auto-narration.

The Department of Defense (DoD) uses modeling and simulation for a variety of purposes, such as to conduct joint training exercises, develop and evaluate new doctrine and tactics, analyze alternative force structures, and study the effectiveness of new weapons systems. Advances in information technology have lowered the cost of computer-based models and simulation, making modeling and simulation a cost-effective alternative to live training and exercises. While these advances have gone a long way towards creating technically accurate simulations they have not addressed the issue of presenting realistic scenarios while supporting user interaction.

The goal of interactive simulation, whether it is a virtual story or a combat simulation, is to present the user with an experience that suspends their disbelief in the artificialities imposed by the system. In this way, the user feels it is a "real" experience. From the DoD perspective, this results in more realistic and effective training, as well as more accurate assessments of the systems, tactics or doctrine being evaluated.

The entertainment industry has long known that to achieve this suspension of disbelief, it is not sufficient to simply produce a technically accurate simulation. It is the unfolding of events and presentation of the story, along with rich believable characters that makes for a truly effective and immersive experience. The Computer-Generated Autonomy Group is exploring the use of autonomous agent technology to guide the behavior of the simulation characters, while constructing a dynamic, interactive story line that is free to unfold based on the actions of the user, the internal states of the autonomous characters, the laws of the simulation world and the global state of the simulation environment.

A system capable of controlling the actions of autonomous computer generated characters within the guidelines of a story or simulation scenario must support complicated worlds with multiple characters and rich plot complications. At the same time, it must be adaptable to multiple domains, whether it be presenting training scenarios in a ground combat simulation or immersing the user in an action-adventure story.

Current approaches based on artificial intelligence planning techniques can support complicated plots with a diverse set of story characters, but they are extremely domain-knowledge specific. Extensive time and effort is required to generate new knowledge bases and dependency networks for each new story. Algorithmic approaches using tree or graph structures to store story events provide a domain independent methodology, but for complicated stories, the tractability of these knowledge structures can be overcome by the combinatorial problem of evaluating all possible plots each time an event occurs [6]. The problem of creating a general interactive story system is one of developing an architecture that scales well and is domain independent.

The Computer-Generated Autonomy Group has developed an interactive, agent-based story system based strongly on the use of *tickets* and *connectors* to present highly interactive and dynamic stories. A typical story consists of goal driven autonomous characters, a narrative structure aligned closely with the protagonist, and a collection of potential scenes, along with media, dialog and character interactions to populate the scenes. These story elements are combined dynamically at runtime to generate a story that adapts to the participants interaction and the state of the participant's character [7].

Figure 5.1 is a screenshot of a scene in which two autonomous characters are conversing in front of a building. The selection of the specific scene within the context of the story is non-scripted. A stage manager agent selects the scene to be played based on many different criteria. Some of these include the protagonist's personality, what the protagonist has experienced thus far in the story, and where the story is with regards to its progression through its narrative phases. Likewise, the interactions between the two characters as the scene plays out, and the consequences of those interactions, are non-scripted. The story is in essence self-organizing, built from the bottom up from a pool of story elements. By taking a bottom up approach, the system is able to overcome the scaling and complexity problems of traditional AI based methods while supporting domain independent story content.

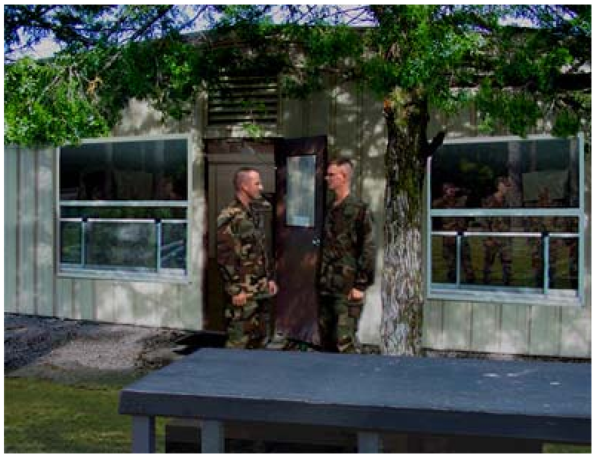

Figure 5.1 Two autonomous characters conversing

5.2. Agent-Based Simulation Auto-Narration

One of the most exciting research projects currently underway is an agent based simulation auto-narrator. When watching MAS simulation demonstrations with dots moving about a screen, a human narrator describes what the dots are doing. But is this interpretation and narration of the agent actions coming from the narrator or from the model? Until the models narrate their own behavior there is no way to know. Through the use of self-documenting *connectors*, analysts will not only be able to study behavior in terms of "what" happened, but the models themselves will provide insight as to "why" it happened.

6. Conclusion

Multi-agent systems (MAS) simulation and autonomous behavior have tremendous potential for application in defense and entertainment/defense projects. The Computer Generated Autonomy Group has made tremendous progress in bringing MAS simulation techniques to Department of Defense (DoD) models and simulations, and advancing the start-of-the-art to make adaptive behavior far easier to create and control. Research projects in helicopter reconnaissance [8], land combat [9], cognitive modeling of land navigation [10], modeling organizational changes in military units [11], naval planning, [12], personnel management [13], human behavior modeling [14], and networked virtual environments [15] have provided valuable insight into their respective problem domains and been well received by their DoD sponsors.

But this work is just the beginning. In the not too distant future, the methodology and tools for creating MAS simulations will be as accessible as those currently available for traditional discrete-event simulations.

References

[1] Hiles, J., VanPutte, M., Osborn, B., Zyda, M., "Innovations in Computer Generated Autonomy at the MOVES Institute", Technical Report NPS-MV-02-002, Naval Postgraduate School, Monterey, California, 2001.

[2] U.S. Department of Defense Directive S-3600-1, 1996 cited in Joint Chiefs of Staff, "Information Assurance: Legal, Regulatory, Policy, and Organizational Considerations", 3rd Edition, U.S. Department of Defense. September 17, 1997.

[3] VanPutte, M., "A Computational Model and Multi-Agent Simulation of Information Assurance", Dissertation, Naval Postgraduate School, Monterey, CA, 2002.

[4] Roddy, K. and Dickson, M., "Modeling Human And Organizational Behavior Using A Relation-Centric Multi-Agent System Design Paradigm", Master's thesis, Naval Postgraduate School, Monterey, CA, 2000.

[5] Kang, M., Waisel, L.B., Wallace, W.A. "Team Soar – A model for team decision Making", Simulating Organizations, AAAI Press, 1998.

[6] Weyhrauch, P., "Guiding Interactive Drama". Ph.D. thesis, Technical Report CMU-CS-97-109, School of Computer Science, Carnegie Mellon University, 1997.

[7] Osborn, B., "An Agent-Based Architecture for Computer-Generated Interactive Stories", Dissertation, Naval Postgraduate School, Monterey, CA, 2002.

[8] Unrath, C., "Dynamic Exploration of Helicopter Reconnaissance Through Agent-based Modeling", Master's thesis, Naval Postgraduate School, Monterey, CA, 2000.

[9] Mert, E. and Jilson, E., "Modeling Conventional Land Combat in a Multi-Agent System Using Generalization of the Different Combat Entities and Combat Operations", Master's thesis, Naval Postgraduate School, Monterey, CA, 2001.

[10] Stine, J., Representing Tactical Land Navigation Expertise, Master's thesis, Naval Postgraduate School, Monterey, CA, 2000.

[11] Pawloski, J., "Modeling Tactical Land Combat Using a Multi-Agent System Design Paradigm (GI Agent)", Master's thesis, Naval Postgraduate School, Monterey, CA, 2001.

[12] Ercetin, A., "Operation-Level Naval Planning Using Agent-Based Simulation", Master's thesis, Naval Postgraduate School, Monterey, CA, 2001.

[13] French, S., "Analyzing Projected Personnel Retention Utilizing Complex Adaptive Systems", Master's thesis, Naval Postgraduate School, Monterey, CA, 2001.

[14] Hennings, C., "Designing Realistic Human Behavior into Multi-Agent Systems", Master's thesis, Naval Postgraduate School, Monterey, CA, 2001.

[15] Washington, D., "Implementation of a Multi-Agent Simulation for the NPSNET-V Virtual Environment Research Project", Master's thesis, Naval Postgraduate School, Monterey, CA, 2001.

Author Biographies

MAJOR MICHAEL VANPUTTE is a Ph.D. candidate in the Computer Science Department, Naval Postgraduate School and a member of the MOVES Institute. His dissertation "A Computational Model and Multi-Agent Simulation of Information Assurance" involved the development of a mathematical and descriptive model of Information Assurance, and a multi-agent implementation of these models. Major VanPutte is an active-duty combat engineer officer. He received his BS in Information Systems from the Ohio State University and MS in Computer Science from the University of Missouri-Columbia. His current interests are multi-agent systems, information assurance, and computer-generated autonomous behavior.

COMMANDER BRIAN OSBORN is Ph.D. candidate in the Computer Science Department of the Naval Postgraduate School and a researcher in the MOVES Institute. His concentrations include agent-based modeling and simulation, computer generated autonomous behavior and interactive narrative. He is the principle implementer of the MOVES Story Engine, developed with the guidance of Professor John Hiles. His dissertation project "An Agent-Based Architecture for Computer-Generated Interactive Stories" explores the use of autonomous agent technology to guide the behavior of the simulation characters, while constructing a dynamic, interactive story line based on the actions of the user, the internal states of the autonomous characters and the laws of the simulation environment. He is a principal investigator in the MOVES Center for the Study of Potential Outcomes where the MOVES Story Engine and agent-based simulation techniques are being applied to the problem of anticipating unexpected actions on the part of systems and organizations such as terrorist groups. CDR Osborn is an active duty Naval aviator. He holds a BA in Applied Mathematics (U Maine, Orono) and a MS in Operations Research from the Naval Postgraduate School.

PROFESSOR JOHN HILES is a research professor in computer science at the Naval Postgraduate School, and a program manager for the MOVES Institute's Army Game Project. Professor Hiles's interests include multi-agent systems, complex adaptive system models and simulations, and the software-systems implications of the biochemistry of the living cell. Professor Hiles has been an advisor to the Marine Corps and Army. He is a frequent participant in the Highlands Forum and an advisor to the Bios Group. In 1992, Professor Hiles began work with adaptive, agent-based models and simulations at Maxis, the company responsible for SimCity. He was first to apply the striking characteristics of SimCity in games/systems designed for government and business. His seven-year focus on simulations such as SimHealth, TeleSim, and Project Challenge refined and extended the use of adaptive, agent-based methods and computer-game human interfaces in commercial products.